UNIVERSIDAD DE CASTILLA LA-MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INTELIGENCIA ARTIFICIAL E INGENIERÍA DEL CONOCIMIENTO:

PRÁCTICA 2: EL JUEGO DE LA BANDERA CON ADVERSARIO - **FINAL**

Alumno: Miguel García Corchero

27 de Mayo de 2009

Índice de contenido

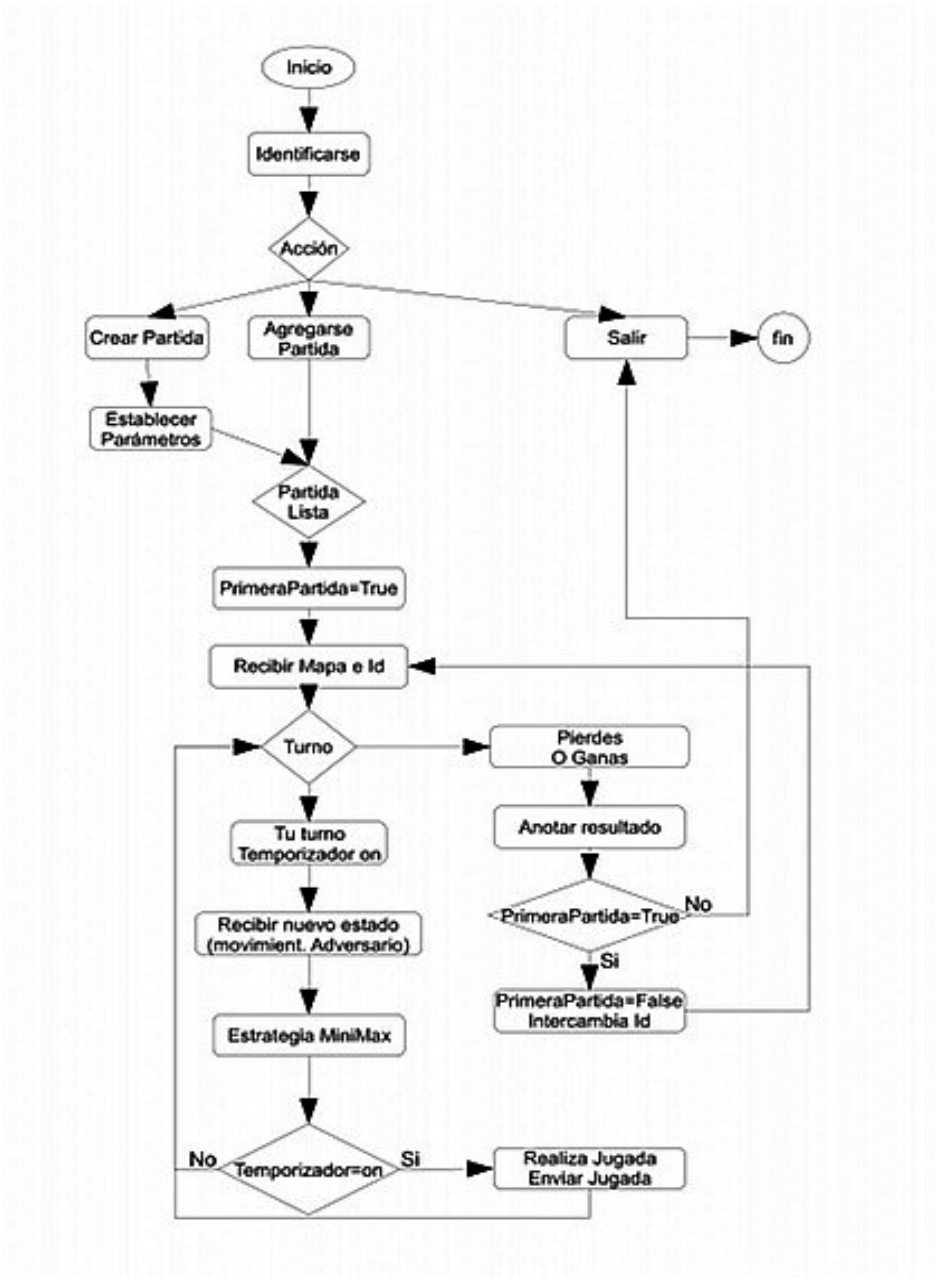
1. Introducción....4
1.1. Definición del problema a tratar....4
1.2. Decisiones de diseño....5
1.3. Estructuras para su resolución....6
2. Soluciones teóricas....7
2.1. Breve explicación de los algoritmos utilizados, poda Alfa-Beta....7
2.2. Heurísticas diseñadas....8
3. Pruebas....9
4. Conclusiones....12
5. Manual de usuario....13
5.1. Configuración....13
5.2. Ejecución de la práctica....13
6.Código Fuente....14
6.1. Makefile....14
6.2. Banderas.sh....14
6.3. Clases.h....15
6.4. Clases.cpp....16
6.5. Client.h....22
6.6. Client.cpp....23
6.7. Problema.h....26
6.8. Problema.cpp....27
6.9. Colores.h....32
6.10. Colores.cpp....32

1. Introducción

1.1. Definición del problema a tratar

Realizar un bot (agente racional) que pueda jugar de forma autónoma contra un adversario, permitiendo la posibilidad de realizar una pequeña competición. La estrategia de juego será dirigida por un algoritmo Mini-Max con poda alfa-beta.

El juego se desarrolla con las reglas establecidas en el anexo "Servidor del Juego de la Bandera". El flujo de control aproximado se muestra en la figura siguiente:



1.2. Decisiones de diseño

El lenguaje elegido para implementar la solución es C++ por su paradigma orientado a objetos y por que genera un ejecutable más rápido y robusto que en otros lenguajes.

A grandes rasgos se han dispuesto los siguientes mecanismos:

- Se crea un hilo donde se lanza la búsqueda mediante algoritmo MiniMax con poda Alfa-Beta usando una búsqueda en profundidad iterativa.

- Cuando se acaba el tiempo abortamos la ejecución de dicho hilo alternativo y lo que tenemos es la última solución buena calculada, que al haber hecho una búsqueda en profundidad iterativa habremos llegado a la máxima profundidad para ese tiempo dado sin correr riesgos.

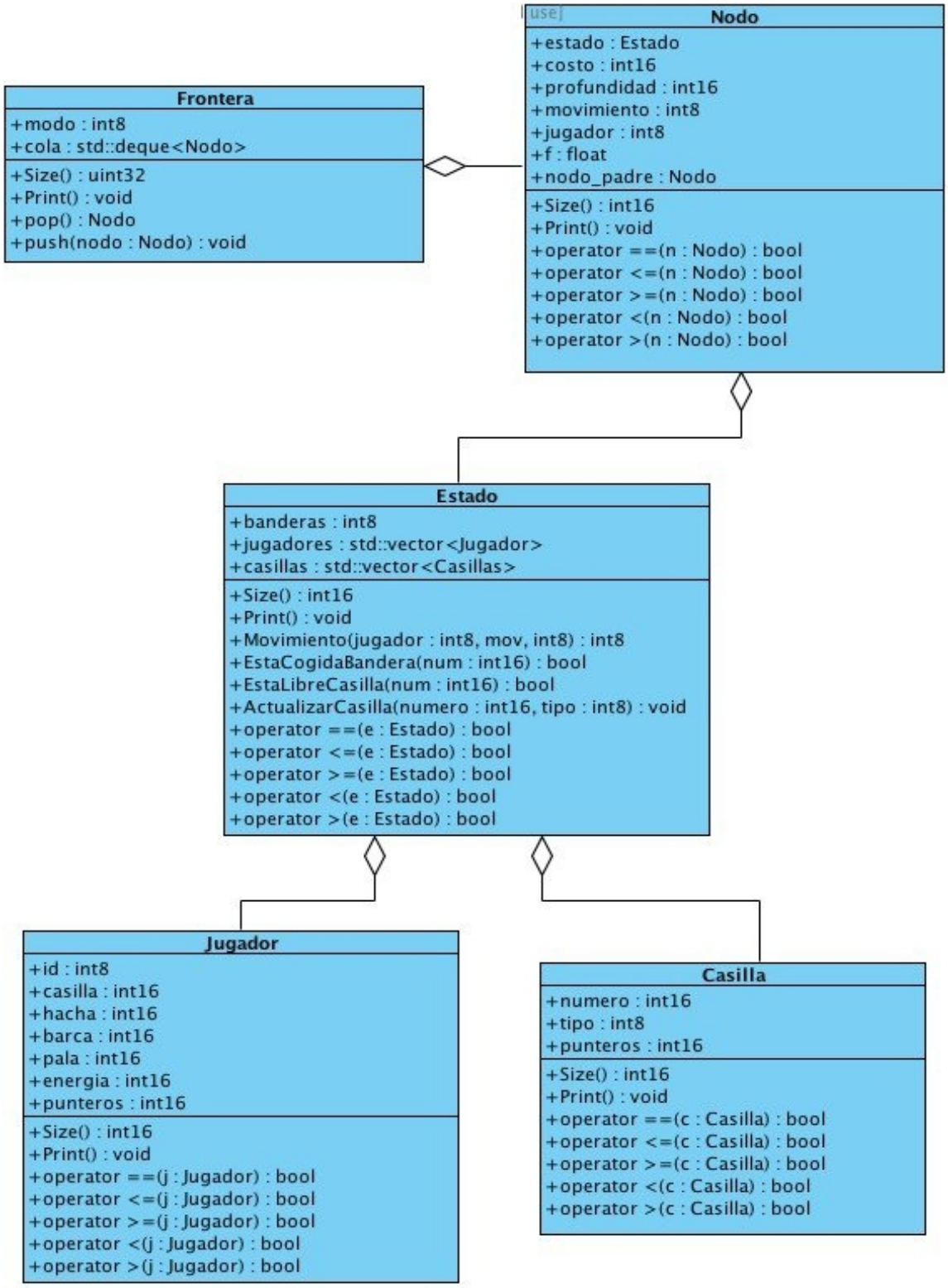
- En cuanto a la búsqueda mediante el algoritmo MiniMax con poda Alfa-Beta se ha optado por implementar una mejora que es la siguiente:

Dada una decisión a tomar, en el último nivel, si tenemos queremos quedarnos con el nodo que tenga el máximo valor MiniMax pero hay varios con dicho valor, entonces, de entre todos esos nodos nos quedaremos con el que mejor resultado de para la función heurística Es decir, para valores iguales, como el valor que han elevado los nodos inferiores es el mismo para todos ellos, calcularemos cual es la valoración de ese nodo y entonces tomaremos la decisión en base a eso.

En cierto modo es como aplicar una búsqueda mediante ascensión a colinas cuando tenemos empate de valores en nodos y no sabemos cual elegir.

1.3. Estructuras para su resolución

Se ha dispuesto el siguiente diagrama de clases UML para la resolución del problema:



2. Soluciones teóricas

2.1. Breve explicación de los algoritmos utilizados, poda Alfa-Beta

Minimax:

Es un método de decisión para minimizar la pérdida máxima esperada en juegos con adversario y con información perfecta. Minimax es un algoritmo recursivo.

El funcionamiento de Minimax puede resumirse como elegir el mejor movimiento para ti mismo suponiendo que tu contrincante escogerá el peor para ti.

Pasos del algoritmo Minimax:

1. Generación del árbol de juego. Se generarán todos los nodos hasta llegar a un estado terminal.
2. Cálculo de los valores de la función de utilidad para cada nodo terminal.
3. Calcular el valor de los nodos superiores a partir del valor de los inferiores. Alternativamente se elegirán los valores mínimos y máximos representando los movimientos del jugador y del oponente, de ahí el nombre de Minimax.
4. Elegir la jugada valorando los valores que han llegado al nivel superior.

El algoritmo explorará los nodos del árbol asignándoles un valor numérico mediante una función de utilidad, empezando por los nodos terminales y subiendo hacia la raíz. La función de utilidad definirá lo buena que es la posición para un jugador cuando la alcanza. En el caso del ajedrez los posibles valores son (+1,0,-1) que se corresponden con ganar, empatar y perder respectivamente. En el caso del backgammon los posibles valores tendrán un rango de [+192,-192], correspondiéndose con el valor de las fichas. Para cada juego pueden ser diferentes.

Poda Alfa-Beta:

La poda Alfa-Beta se basa en la idea de disponer de dos valores que conforman una ventana a la cual deben pertenecer los valores de la función eval(n) para que sean considerados. En los nodos n MAX, según el algoritmo minimax, se debe maximizar el valor de los nodos sucesores. En estos nodos, se utiliza el parámetro a (n) que determina el máximo de los valores de los nodos sucesores encontrado hasta el momento. Así mismo, como los nodos n MIN tratan de minimizar el valor de los nodos, utilizan el parámetro b (n) que va a ser, en cada momento, el mínimo de los valores encontrados de los nodos sucesores de ese nodo.

Existen dos formas de podar, dependiendo del nodo en el que se está estudiando. En los nodos MAX, la condición de poda es: a p >= b p-1 es decir, si el a de un nodo MAX de profundidad p es mayor o igual que el b del nodo MIN padre (profundidad p-1) se pueden podar los sucesores del nodo MAX no estudiados todavía. Esto es debido a que, como valor inferior, el nodo MAX va a

devolver ese valor de a (a p). Ya que el nodo superior trata de minimizar, va a calcular:

$$\beta_{p-1} = \begin{cases} \beta_{p-1} & \text{si } \alpha_p >= \beta_{p-1} \\ \alpha_p & \text{si } \alpha_p < \beta_{p-1} \end{cases}$$

con lo que siempre va a elegir el b_{p-1}. Por lo tanto, los nodos no estudiados todavía en el nodo MAX no es necesario estudiarlos, porque no cambian el valor b del nodo padre.

Simétricamente, la condición de poda en los nodos MIN es:$b_p <= a_{p-1.}$ La explicación es análoga al caso anterior. El algoritmo recursivo del Alfa-Beta podría especificarse de la forma expresada en la tabla adjunta, donde la llamada inicial es de la forma: Alfa-Beta (Situación Inicial Menor-Número Mayor-Número 0).

2.2. Heurísticas diseñadas

La heurística diseñada es una función de la forma:

Eval(s)=w1f1(s) + w2f2(s) + w3f3(s) + w4f4(s)

Donde :

w1= peso del factor 1 = 0.7
w1= peso del factor 2 = 0.1
w1= peso del factor 3 = 0.1
w1= peso del factor 4 = 0.1

f1(s)= suma de las distancias a las banderas
f2(s)= preferencia por estados con casillas de jugador de menos costo
f3(s)= numero de banderas conseguidas
f4(s)= profundidad de ese nodo

3. Pruebas

El equipo sobre el que se han realizado las pruebas es el siguiente:

Software:

SO: Ubuntu Gnu/Linux 8.10
Kernel 2.6.27-7
Versión 64 Bits

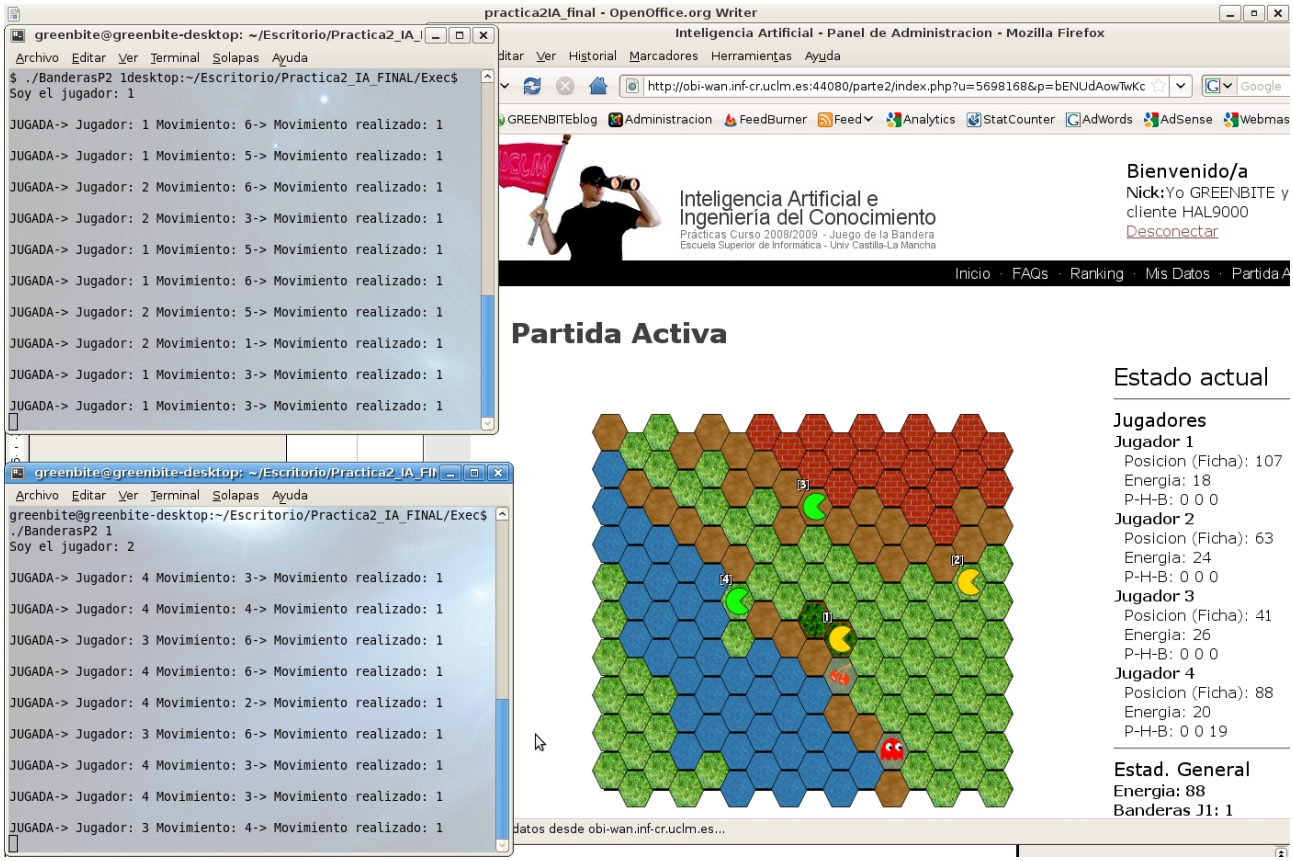
Lenguaje de programación: C++ & STL & ICE Runtime

Hardware:

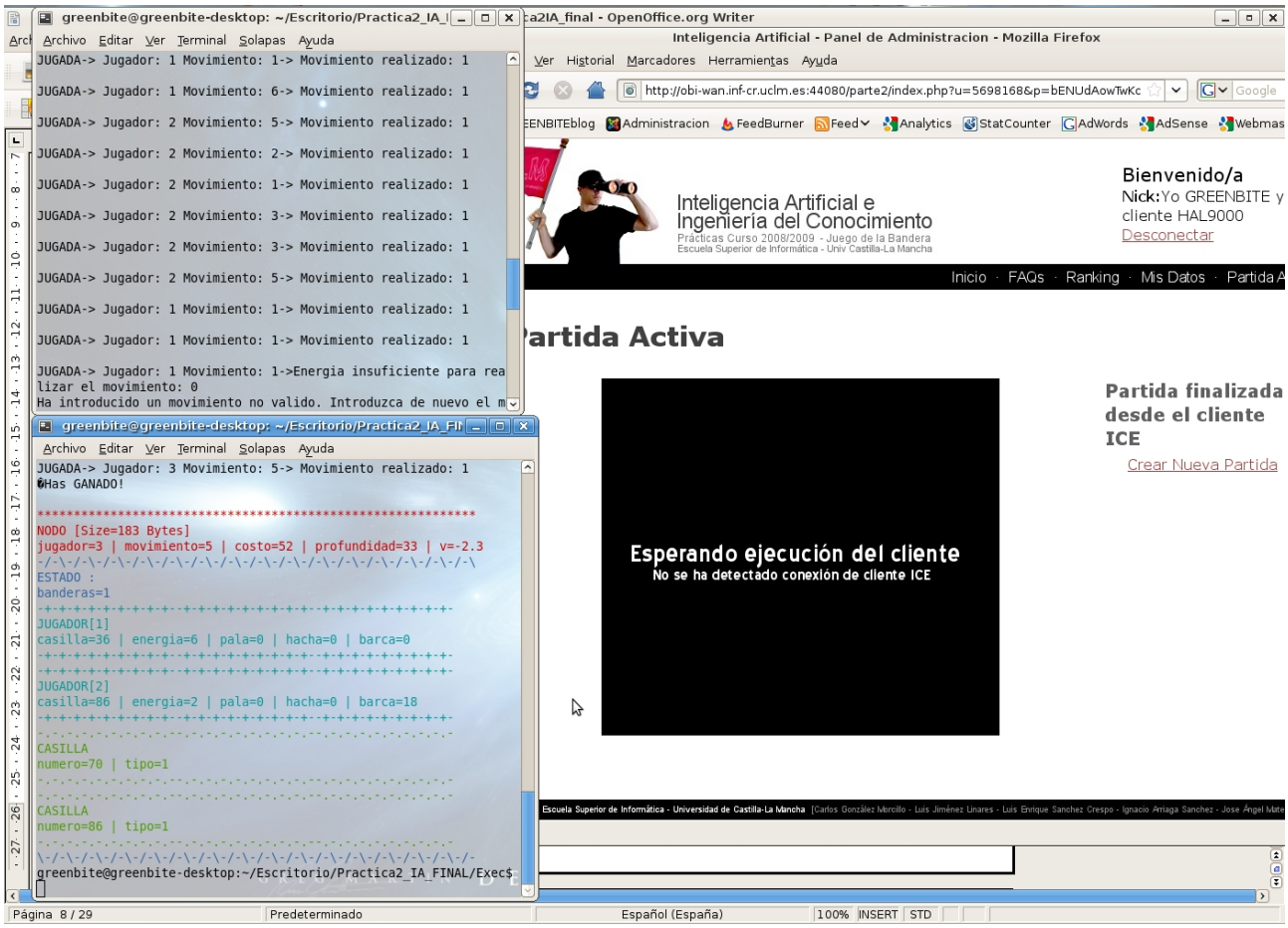
Procesador: Intel Quad Core 2.4 Ghz
Memoria: 4 Gb – 800 Mhz

En una partida con un mapa normal se ha encontrado que llega a profundidad 8.

A continuación una captura del juego mientras se está jugando:


practica2IA_final - OpenOffice.org Writer
greenbite@greenbite-desktop: ~/Escritorio/Practica2_IA_I
Archivo Editar Ver Terminal Solapas Ayuda
$ ./BanderasP2 1desktop:~/Escritorio/Practica2_IA_FINAL/Exec$
Soy el jugador: 1
JUGADA-> Jugador: 1 Movimiento: 6-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 5-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 6-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 3-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 5-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 6-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 5-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 1-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 3-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 3-> Movimiento realizado: 1
greenbite@greenbite-desktop: ~/Escritorio/Practica2_IA_FI
Archivo Editar Ver Terminal Solapas Ayuda
greenbite@greenbite-desktop:~/Escritorio/Practica2_IA_FINAL/Exec$
./BanderasP2 1
Soy el jugador: 2
JUGADA-> Jugador: 4 Movimiento: 3-> Movimiento realizado: 1
JUGADA-> Jugador: 4 Movimiento: 4-> Movimiento realizado: 1
JUGADA-> Jugador: 3 Movimiento: 6-> Movimiento realizado: 1
JUGADA-> Jugador: 4 Movimiento: 6-> Movimiento realizado: 1
JUGADA-> Jugador: 4 Movimiento: 2-> Movimiento realizado: 1
JUGADA-> Jugador: 3 Movimiento: 6-> Movimiento realizado: 1
JUGADA-> Jugador: 4 Movimiento: 3-> Movimiento realizado: 1
JUGADA-> Jugador: 4 Movimiento: 3-> Movimiento realizado: 1
JUGADA-> Jugador: 3 Movimiento: 4-> Movimiento realizado: 1
Inteligencia Artificial - Panel de Administracion - Mozilla Firefox
http://obi-wan.inf-cr.uclm.es:44080/parte2/index.php?u=5698168&p=bENUdAowTwKc
Inteligencia Artificial e Ingeniería del Conocimiento
Bienvenido/a
Nick:Yo GREENBITE y
cliente HAL9000
Desconectar
Inicio · FAQs · Ranking · Mis Datos · Partida A
Partida Activa
Estado actual
Jugadores
Jugador 1
Posicion (Ficha): 107
Energia: 18
P-H-B: 0 0 0
Jugador 2
Posicion (Ficha): 63
Energia: 24
P-H-B: 0 0 0
Jugador 3
Posicion (Ficha): 41
Energia: 26
P-H-B: 0 0 0
Jugador 4
Posicion (Ficha): 88
Energia: 20
P-H-B: 0 0 19
Estad. General
Energia: 88
Banderas J1: 1

Y la conclusión del mismo, con un ganador:


greenbite@greenbite-desktop: ~/Escritorio/Practica2_IA_
Archivo Editar Ver Terminal Solapas Ayuda
JUGADA-> Jugador: 1 Movimiento: 1-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 6-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 5-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 2-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 1-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 3-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 3-> Movimiento realizado: 1
JUGADA-> Jugador: 2 Movimiento: 5-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 1-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 1-> Movimiento realizado: 1
JUGADA-> Jugador: 1 Movimiento: 1->Energia insuficiente para rea
lizar el movimiento: 0
Ha introducido un movimiento no valido. Introduzca de nuevo el m
greenbite@greenbite-desktop: ~/Escritorio/Practica2_IA_FI
Archivo Editar Ver Terminal Solapas Ayuda
JUGADA-> Jugador: 3 Movimiento: 5-> Movimiento realizado: 1
HAs GANADO!
NODO [Size=183 Bytes]
jugador=3 | movimiento=5 | costo=52 | profundidad=33 | v=-2.3
ESTADO :
banderas=1
JUGADOR[1]
casilla=36 | energia=6 | pala=0 | hacha=0 | barca=0
JUGADOR[2]
casilla=86 | energia=2 | pala=0 | hacha=0 | barca=18
CASILLA
numero=70 | tipo=1
CASILLA
numero=86 | tipo=1
greenbite@greenbite-desktop:~/Escritorio/Practica2_IA_FINAL/Exec$
a2IA_final - OpenOffice.org Writer
Inteligencia Artificial - Panel de Administracion - Mozilla Firefox
Ver Historial Marcadores Herramientas Ayuda
http://obi-wan.inf-cr.uclm.es:44080/parte2/index.php?u=5698168&p=bENUdAowTwKc
Administracion FeedBurner Feed Analytics StatCounter AdWords AdSense Webmas
Inteligencia Artificial e
Ingeniería del Conocimiento
Prácticas Curso 2008/2009 - Juego de la Bandera
Escuela Superior de Informática - Univ Castilla-La Mancha
Bienvenido/a
Nick:Yo GREENBITE y
cliente HAL9000
Desconectar
Inicio · FAQs · Ranking · Mis Datos · Partida A
artida Activa
Esperando ejecución del cliente
No se ha detectado conexión de cliente ICE
Partida finalizada
desde el cliente
ICE
Crear Nueva Partida
Escuela Superior de Informática - Universidad de Castilla-La Mancha
Página 8 / 29
Predeterminado
Español (España)
100%
INSERT
STD

4. Conclusiones

Podemos quedarnos con la idea de que tomar decisiones óptimas en juegos que están limitados en tiempo es una tarea bastante complicada. El algoritmo MiniMax nos permite trabajar con este tipo de problemas y si además utilizamos poda Alfa-Beta podremos bajar mucho más en el árbol de búsqueda, lo que significa calcular movimientos más inteligentes.

Por otro lado el diseño de una heurística acertada y que permita una ejecución rápida nos permitirá también bajar mas aún en el árbol de búsqueda y que nuestro bot inteligente pueda mirar más lejos a la hora de planificar un movimiento.

Como conclusión decir que para la resolución correcta de este tipo de problemas es importante elegir un lenguaje que permita un nivel de optimización alto, pero el punto crítico sin duda es el diseño de una heurística que sea buena.

5. Manual de usuario

5.1. Configuración

Primero debemos compilar el programa, lo cual es bastante sencillo, entramos en el directorio donde está el código fuente y compilamos de la siguiente forma:

$ make

Esto generara un ejecutable, y además tenemos un script de shell (BanderasP2) para hacer más sencillo su uso.

5.2. Ejecución de la práctica

Para jugar una partida individual debemos hacerlo de la siguiente forma:

- Debemos abrir 2 terminales diferentes donde tenemos el ejecutable y ejecutar en cada uno de ellos la siguiente orden:

$ ./BanderasP2 1

Para jugar una partida contra contrincante lo haremos de la siguiente forma:

- Abrimos un único termina donde tenemos el ejecutable y cuando la partida esté creada en el servidor ejecutamos la orden:

$ ./BanderasP2 2

6.Código Fuente

6.1. Makefile

```
CXXFLAGS=-I./ -I/usr/lib/ -lm -lpthread
SLICOMPILER=slice2cpp
#cuando no queramos modo debug dejamos vacío el valor de esta variable
DEBUG=-Wall
OPTIMIZACIONES = -o3 -mmmx -msse -msse2 -msse3
SLICEFLAGS=-I/usr/share/slice/ --output-dir
PROGRAMNAME=Client
EDITOR=gedit
COMPILADOR=g++

all: $(PROGRAMNAME)

$(PROGRAMNAME): Practica.cpp Practica.o Problema.o Clases.o Client.o Colores.o
	$(COMPILADOR) $(OPTIMIZACIONES) $(DEBUG) Practica.o Problema.o Clases.o Client.o Colores.o -o
$(PROGRAMNAME) -lIce $(CXXFLAGS)

Practica.cpp: Practica.ice
	$(SLICOMPILER) Practica.ice $(SLICEFLAGS) ./

Client.o: Client.cpp Client.h
	$(COMPILADOR) $(OPTIMIZACIONES) $(DEBUG) -c Client.cpp -o Client.o $(CXXFLAGS)

Practica.o: Practica.cpp Practica.h
	$(COMPILADOR) $(OPTIMIZACIONES) $(DEBUG) -c Practica.cpp -o Practica.o $(CXXFLAGS)

Problema.o: Problema.cpp Problema.h
	$(COMPILADOR) $(OPTIMIZACIONES) $(DEBUG) -c Problema.cpp -o Problema.o $(CXXFLAGS)

Clases.o: Clases.cpp Clases.h
	$(COMPILADOR) $(OPTIMIZACIONES) $(DEBUG) -c Clases.cpp -o Clases.o $(CXXFLAGS)

Colores.o: Colores.cpp Colores.h
	$(COMPILADOR) $(OPTIMIZACIONES) $(DEBUG) -c Colores.cpp -o Colores.o $(CXXFLAGS)

cleano:
	$(RM) *.o

clean:
	$(RM) $(PROGRAMNAME) *.o Practica.cpp Practica.h

clear:
	$(RM) $(PROGRAMNAME) *.o Practica.cpp Practica.h

edit:
	$(EDITOR) Client.cpp Client.h Problema.cpp Problema.h Clases.cpp Clases.h Colores.cpp
Colores.h Banderas Makefile Configuracion &
config:
	$(EDITOR) Configuracion &
```

6.2. Banderas.sh

```
#!/bin/sh

./Client $1 --Ice.Config=./config.client
```

6.3. Clases.h

```
#ifndef __CLASES_H_
#define __CLASES_H_

#include "Client.h"
#include "Colores.h"
#include <memory>
#include <math.h>

using namespace std;

class Casilla{
	public:
		int16 numero,punteros; int8 tipo;

		Casilla(int16 numero,int8 tipo);
		~Casilla();

		void Print();
		int16 Size();

		bool operator == (Casilla *c);
		bool operator < (Casilla *c);
		bool operator <= (Casilla *c);
		bool operator > (Casilla *c);
		bool operator >= (Casilla *c);
};

class Jugadores{
	public:
		int8 id;
		int16 casilla,hacha,barca,pala,punteros;
		sint16 energia;

		Jugadores(int8 id,int16 casilla,int16 energia,int16 hacha,int16 barca,int16 pala);
		~Jugadores();

		void Print();
		int16 Size();

		bool operator == (Jugadores *j);
		bool operator < (Jugadores *j);
		bool operator > (Jugadores *j);
		bool operator <= (Jugadores *j);
		bool operator >= (Jugadores *j);
};

class Estado{
	public:
		int8 banderas;
		vector<Jugadores*> jugadores;
		vector<Jugadores*> contrarios;
		vector<Casilla*> casillas;

		Estado(int8 banderas,vector<Jugadores*> *jugadores,vector<Casilla*> *casillas);
		~Estado();

		void Print();
		int16 Size();
		bool EstaCogidaBandera(int16 num);
		bool EstaLibreCasilla(int16 num);
		sint8 Movimiento(int8 jugador,int8 mov);
		void ActualizarCasilla(int16 numero,int8 tipo);

		bool operator == (Estado *e);
		bool operator < (Estado *e);
		bool operator > (Estado *e);
		bool operator <= (Estado *e);
		bool operator >= (Estado *e);
};

class Nodo{
	public:
		Estado *estado;
		int16 costo,profundidad;
```

```
                int8 movimiento,jugador;
                Nodo *nodo_padre;
                float v; //valor de utilidad

                Nodo(Estado *estado,Nodo *nodo_padre,int8 jugador,int8 movimiento,int16 costo,int16
profundidad,float v);
                ~Nodo();

                void Print();
                void Copiar(Nodo *nodo);
                int16 Size();

                bool operator ==(Nodo *n);
                bool operator !=(Nodo *n);
                bool operator <(Nodo *n);
                bool operator <=(Nodo *n);
                bool operator >(Nodo *n);
                bool operator >=(Nodo *n);
};

class Frontera{
        private:
                deque<Nodo*> *c; int8 modo;
        public:
                Frontera(int8 modo);
                ~Frontera();
                Nodo* pop();
                void push(Nodo* nodo);

                unsigned int size();
                void Print();
};
#endif
```

6.4. Clases.cpp

```
#include "Clases.h"
#include "Problema.h"

extern Mapa mapa;
extern int8 njugadores;
#define round(x) (x<0?ceil((x)-0.5):floor((x)+0.5))
//********************************************************************************************
//CASILLA
//********************************************************************************************
Casilla::Casilla (int16 numero,int8 tipo){ this->numero=numero; this->tipo=tipo; punteros=1; }
Casilla::~Casilla(){};
void Casilla::Print(){
        color_verde();
        cout << endl << "-.-.-.-.-.-.-.-.-.-.--.-.-.-.-.-.-.-.-.--.-.-.-.-.-.-.-.-";
        cout << endl << "CASILLA";
        cout << endl << "numero=" << (int)(numero+1) << " | tipo=" << (int)tipo;
        cout << endl << "-.-.-.-.-.-.-.-.-.-.--.-.-.-.-.-.-.-.-.--.-.-.-.-.-.-.-.-";
        color_normal();
}
int16 Casilla::Size(){ return (sizeof(int8) + (sizeof(int16)*2) + sizeof(Casilla)); }
bool Casilla::operator ==(Casilla *c){
        if ((numero == c->numero) && (tipo != c->tipo)) { return true; }
        else { return false; }
}
bool Casilla::operator <(Casilla *c){
        if (this->numero < c->numero) { return true; }
        else { return false; }
}
bool Casilla::operator <=(Casilla *c){
        if (this->numero < c->numero) { return false; }
        else { return true; }
}
bool Casilla::operator >(Casilla *c){
        if (this->numero > c->numero) { return true; }
        else { return false; }
}
bool Casilla::operator >=(Casilla *c){
        if (this->numero < c->numero) { return false; }
        else { return true; }
}
```

```
//*********************************************************************************************
//NODO
//*********************************************************************************************
Nodo::Nodo(Estado *estado,Nodo *nodo_padre,int8 jugador,int8 movimiento,int16 costo,int16
profundidad,float v){
               this->estado=estado; this->nodo_padre=nodo_padre; this->jugador=jugador; this->v=v;
               this->costo=costo; this->profundidad=profundidad; this->movimiento=movimiento;
}
Nodo::~Nodo(){ nodo_padre=NULL; delete estado; }
void Nodo::Print(){
        color_rojo(); cout << endl << "************************************************************";
        cout << endl << "NODO [Size=" << (int)Size() << " Bytes]";
        cout << endl << "jugador=" << (int)jugador << " | movimiento=" << (int)movimiento << " |
costo=" << (int)costo << " | profundidad=" << (int)profundidad  << " | v=" << (float)v;
        estado->Print();
        color_rojo(); cout << endl << "************************************************************";
color_normal();
}
int16 Nodo::Size(){ return (estado->Size() + (sizeof(int16)*2) + (sizeof(int8)*2) + sizeof(Nodo) +
sizeof(Nodo*) + sizeof(Estado*) ) ; }
bool Nodo::operator ==(Nodo *n){
        if (this->profundidad != n->profundidad){ return false; }
        else if (this->costo != n->costo) { return false; }
        else if (this->movimiento != n->movimiento) { return false; }
        else if (this->jugador != n->jugador) { return false; }
        else if (!(this->estado == n->estado)) { return false; }
        else { return true; }
}
//Cuidado, usados para ordenar A*
bool Nodo::operator !=(Nodo *n){
        return (!(this->v == n->v));
}
bool Nodo::operator <=(Nodo *n){
        if (this->v <= n->v) { return true; }
        else { return false; }
}
bool Nodo::operator >=(Nodo *n){
        if (this->v >= n->v) { return true; }
        else { return false; }
}
bool Nodo::operator >(Nodo *n){
        if (this->v > n->v) { return true; }
        else { return false; }
}
bool Nodo::operator <(Nodo *n){
        if (this->v < n->v) { return true; }
        else { return false; }
}
//*********************************************************************************************
//ESTADO
//*********************************************************************************************
Estado::Estado(int8 banderas,vector<Jugadores*> *jugadores,vector<Casilla*> *casillas){
        this->jugadores.reserve(jugadores->size()); this->casillas.reserve(casillas->size());
        this->banderas=banderas;
        for (int8 k=0;k<(int8)(jugadores->size());k++){ (*jugadores)[k]->punteros++; this-
>jugadores.push_back((*jugadores)[k]); }
        for (int16 k=0;k<(int16)(casillas->size());k++){ (*casillas)[k]->punteros++; this-
>casillas.push_back((*casillas)[k]); }
}
Estado::~Estado(){
        for(int8 i = 0; i < (int8)jugadores.size(); i++) {
                try {
                        std::vector<Jugadores*>::iterator p;
                        for (p=jugadores.begin();p<jugadores.end();p++){
                                jugadores.erase(p);
                        }
                 } catch( char * str ) {       cout << endl << "Exception de borrado de Jugador: " <<
str << endl; }
        }
        for(int16 i = 0; i < (int16)casillas.size();  i++) {
                try {
                        std::vector<Casilla*>::iterator p;
                        for (p=casillas.begin();p<casillas.end();p++){
                                casillas.erase(p);
                        }
                 } catch( char * str ) {       cout << endl << "Exception de borrado de Casilla: " <<
str << endl; }
        }
```

```
        jugadores.clear(); casillas.clear();
}
void Estado::Print(){
        color_azul(); cout << endl <<
"-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\";
        cout << endl << "ESTADO : ";
        cout << endl << "banderas=" << (int)banderas;
        for (int8 i=0;i<(int8)jugadores.size()/2;i++){ jugadores[i]->Print(); }
        for (int16 i=0;i<(int16)casillas.size();i++){ casillas[i]->Print(); }
        color_azul(); cout << endl <<
"\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-\\-/-"; color_normal();
}
int16 Estado::Size(){
        return ( (sizeof(int8) + sizeof(jugadores) + sizeof(casillas) + sizeof(Estado) ) );
}
bool Estado::EstaCogidaBandera(int16 num){
        for(int16 i = 0; i < (int16)casillas.size(); i++) {
                if ( casillas[i]->numero == num ) { return true; }
        }
        return false;
}
bool Estado::EstaLibreCasilla(int16 num){
        bool resultado=true;
        for (int16 i=0;i<(int16)jugadores.size()/2;i++){
                if (jugadores[i]->casilla==num){ resultado=false; break; }
        }
        return resultado;
}
void Estado::ActualizarCasilla(int16 numero,int8 tipo){
        int16 i; bool actualizado=false;

        for (i=0;i<(int16)casillas.size();i++){
                        if (casillas[i]->numero==numero) {
                                if (casillas[i]->tipo!=tipo) {
                                        Casilla *casilla_copia;
                                        try {
                                                casilla_copia=new Casilla(numero,tipo);

                                        } catch (std::bad_alloc) {
                                                cout << endl << "No hay disponible memoria para crear
más instancias de Casilla";
                                                exit(1);
                                        }
                                        std::vector<Casilla*>::iterator posicion=casillas.begin()+i;
                                        (*posicion)->punteros-=1;
                                        casillas.erase(posicion);
casillas.insert(posicion,casilla_copia);
                                } actualizado=true; break;
                        }
        }

        if (!actualizado){
                Casilla *c;
                try {
                        c = new Casilla(numero,tipo); casillas.push_back(c);
                } catch (std::bad_alloc) {
                        cout << endl << "No hay disponible memoria para crear más instancias de
Casilla";
                        exit(1);
                }
        }
}
sint8 Estado::Movimiento (int8 jugador,int8 mov){
        Jugadores *jugador_copia;
        try {
                jugador_copia=new Jugadores(
                           jugadores[jugador]->id,
                           jugadores[jugador]->casilla,
                           jugadores[jugador]->energia,
                           jugadores[jugador]->hacha,
                           jugadores[jugador]->barca,
                           jugadores[jugador]->pala);
        } catch (std::bad_alloc) {
                cout << endl << "No hay disponible memoria para crear más instancias de Jugador";
                exit(1);
        }

        std::vector<Jugadores*>::iterator posicion=jugadores.begin()+jugador;
        (*posicion)->punteros-=1;
```

```
        jugadores.erase(posicion); jugadores.insert(posicion,jugador_copia);

        int16 nueva_casilla=SiguienteCasilla(jugadores[jugador]->casilla,mov),i;
        sint8 resultado=0; sint8 tipo_casilla=0;

        if (EstaLibreCasilla(nueva_casilla)){
                //obtenemos el valor de las casillas actualizadas
                for (i=0;i<(int16)casillas.size();i++){
                        if (casillas[i]->numero==(nueva_casilla-1)) { tipo_casilla=casillas[i]-
>tipo; }
                }

                //si no ha sido actualizada obtenemos el tipo de casilla del mapa original
                if (tipo_casilla==0) { tipo_casilla=mapa.casillas[nueva_casilla-1]; }

                switch (tipo_casilla){
                        case HIERBA:   jugadores[jugador]->energia--; break;
                        case AGUA:     if (jugadores[jugador]->barca>0) {jugadores[jugador]->energia-
=3; jugadores[jugador]->barca--;}
                                       else { jugadores[jugador]->energia-=6; } break;
                        case BARRO:    jugadores[jugador]->energia-=2; break;
                        case HOYO:     jugadores[jugador]->energia-=4; break;
                        case ZANJA:    jugadores[jugador]->energia-=6; break;
                        case BANDERA:  resultado=1; ActualizarCasilla(nueva_casilla-1,HIERBA); break;
                        case BARCA:    ActualizarCasilla(nueva_casilla-1,HIERBA); jugadores[jugador]-
>barca+=20; break;
                        case HACHA:    ActualizarCasilla(nueva_casilla-1,HIERBA); jugadores[jugador]-
>hacha+=20; break;
                        case ZUMO:     ActualizarCasilla(nueva_casilla-1,HIERBA); jugadores[jugador]-
>energia+=20; break;
                        case PALA:     ActualizarCasilla(nueva_casilla-1,HIERBA); jugadores[jugador]-
>pala+=10; break;
                        case BOSQUE:   if (jugadores[jugador]->hacha>0) { jugadores[jugador]->energia-
=4; jugadores[jugador]->hacha--; }
                                       else { jugadores[jugador]->energia-=8; } break;
                        case MURAYA:   resultado=ERROR; break;
                }
                if (jugadores[jugador]->energia<=0) { resultado=ERROR; }
                jugadores[jugador]->casilla=nueva_casilla;
        } else { resultado=ERROR; }
        return resultado;
}
bool Estado::operator ==(Estado *e){
        if (banderas!= e->banderas) { return false; }

        if (casillas.size() != e->casillas.size() ) { return false; }
        else {
                for(int16 i = 0; i < (int16)casillas.size(); i++) {
                        if (!(casillas[i] == e->casillas[i] )) { return false; }
                }
        }

        if (jugadores.size() != e->jugadores.size()) { return false; }
        else {
                for(int8 i = 0; i < (int8)jugadores.size(); i++) {
                        if (!(jugadores[i] == e->jugadores[i] )) { return false; }
                }
        }
        return true;
}
bool Estado::operator <(Estado *e){
        int16 energia_this=0,energia_e=0; bool resultado=false;
        for(int8 i = 0; i < (int8)jugadores.size(); i++) {  energia_this+=jugadores[i]->energia; }
        for(int8 i = 0; i < (int8)e->jugadores.size(); i++) {  energia_e+=e->jugadores[i]->energia; }
        if (energia_this>energia_e) { resultado=true; } else { resultado=false; }
        return resultado;
}
bool Estado::operator >(Estado *e){
        int16 energia_this=0,energia_e=0; bool resultado=false;
        for(int8 i = 0; i < (int8)jugadores.size(); i++) {  energia_this+=jugadores[i]->energia; }
        for(int8 i = 0; i < (int8)e->jugadores.size(); i++) {  energia_e+=e->jugadores[i]->energia; }
        if (energia_this<energia_e) { resultado=true; } else { resultado=false; }
        return resultado;
}
bool Estado::operator <=(Estado *e){
        int16 energia_this=0,energia_e=0; bool resultado=false;
        for(int8 i = 0; i < (int8)jugadores.size(); i++) {  energia_this+=jugadores[i]->energia; }
        for(int8 i = 0; i < (int8)e->jugadores.size(); i++) {  energia_e+=e->jugadores[i]->energia; }
        if (energia_this>=energia_e) { resultado=true; } else { resultado=false; }
```

```
        return resultado;
}
bool Estado::operator >=(Estado *e){
        int16 energia_this=0,energia_e=0; bool resultado=false;
        for(int8 i = 0; i < (int8)jugadores.size(); i++) {  energia_this+=jugadores[i]->energia; }
        for(int8 i = 0; i < (int8)e->jugadores.size(); i++) {  energia_e+=e->jugadores[i]->energia; }
        if (energia_this<=energia_e) { resultado=true; } else { resultado=false; }
        return resultado;
}
//*********************************************************************************************
//JUGADORES
//*********************************************************************************************
Jugadores::Jugadores(int8 id,int16 casilla,int16 energia,int16 hacha,int16 barca,int16 pala){
        this->id=id; this->casilla=casilla; this->energia=energia;
        this->hacha=hacha; this->barca=barca; this->pala=pala;
        punteros=1;
}
Jugadores::~Jugadores(){ };
void Jugadores::Print(){
        color_cyan();
        cout << endl << "-+-+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+-";
        cout << endl << "JUGADOR[" << (int)id << "]";
        cout << endl << "casilla=" << (int)casilla << " | energia=" << (int)energia << " | pala=" <<
(int)pala << " | hacha=" << (int)hacha << " | barca=" << (int)barca;
        cout << endl << "-+-+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+--+-+-+-+-+-+-+-+-";
        color_normal();
}
int16 Jugadores::Size(){ return ( (sizeof(int16)*6) + sizeof(int8) + sizeof(Jugadores) ); }
bool Jugadores::operator ==(Jugadores *j){
        if ((casilla == j->casilla) && (pala == j->pala) && (hacha == j->barca)) { return true; }
        else {  return false; }
}
bool Jugadores::operator <(Jugadores *j){
        if (id < j->id) { return true; }
        else { return false; }
}
bool Jugadores::operator <=(Jugadores *j){
        if (id > j->id) { return false; }
        else { return true; }
}
bool Jugadores::operator >(Jugadores *j){
        if (id > j->id) { return true; }
        else { return false; }
}
bool Jugadores::operator >=(Jugadores *j){
        if (id < j->id) { return false; }
        else { return true; }
}
//*********************************************************************************************
//FRONTERA
//*********************************************************************************************
Frontera::Frontera(int8 modo){
        try {
                c=new deque<Nodo*>; this->modo=modo;
        } catch (std::bad_alloc) {
                cout << endl << "No hay disponible memoria para crear más instancias de Frontera";
                exit(1);
        }
}
Frontera::~Frontera(){
        for (std::deque<Nodo*>::iterator i=c->begin();i<c->end();i++){ delete (*i); }
        delete c;
}
Nodo *Frontera::pop(){
        Nodo *nodo;
        switch(modo){
                case COLA_FIFO:
                                nodo=c->front();
                                c->pop_front();
                                break;
                case COLA_LIFO:
                                nodo=c->back();
                                c->pop_back();

                                break;
                case COLA_ORDENADA:
                                nodo=c->front();
                                c->pop_front();
                                break;
```

```
        };
        return nodo;
}
void Frontera::push(Nodo* nodo){
        switch(modo){
                case COLA_FIFO:
                                c->push_back(nodo);
                                break;
                case COLA_LIFO:
                                c->push_back(nodo);
                                break;
                case COLA_ORDENADA:
                                { unsigned int size=c->size();
                                if (size>0) {
                                        //primer elemento
                                        if (*nodo <=(c->front())) { c->push_front(nodo); }
                                        //ultimo elemento
                                        else if (*nodo >=(c->back())) { c->push_back(nodo); }
                                        else { //en medio
                                                if (size<CAMBIO_ALGORITMO1){
                                                        //Algoritmo para n pequeño
                                                        std::deque<Nodo*>::iterator i;
                                                        float fup,fdown;
                                                        fup=(c->back()->v)-(nodo->v);
                                                        fdown=(nodo->v)-(c->front()->v);
                                                        if (fdown <= fup){
                                                                i=c->begin(); int j=c->size();
                                                                while ((*nodo >=(*i)) && (j>0) ) { i++;
j--; }
                                                                c->insert(i,nodo);
                                                        } else {
                                                                i=c->end()-1; int j=c->size();
                                                                while ((*nodo <=(*i)) && (j>0) ) { i--;
j--; }
                                                                c->insert(i+1,nodo);
                                                        }
                                                } else {
                                                        //Algoritmo para n grande
                                                        std::deque<Nodo*>::iterator inicio,fin,medio,i;
                                                        inicio=c->begin(); fin=c->end()-1; medio=inicio
+ ((fin-inicio)/2);
                                                        i=medio;
                                                        while (medio!=fin && medio!=inicio && nodo!
=(*i)){
                                                                if (*nodo>*medio){
                                                                        inicio=medio; fin=fin;
medio=inicio + ((fin-inicio)/2);
                                                                } else if (*nodo<*medio){
                                                                        inicio=inicio; fin=medio;
medio=inicio + ((fin-inicio)/2);
                                                                } //igual que medio
                                                                else { i=medio; break; }
                                                                i=medio;
                                                        }
                                                        c->insert(i+1,nodo);
                                                }
                                        }
                                } else { c->push_back(nodo); } }
                                break;
        }
}
void Frontera::Print(){ for (std::deque<Nodo*>::iterator i=c->begin();i<c->end();i++){ (*i)-
>Print(); } }
unsigned int Frontera::size(){ return c->size(); }
```

6.5. Client.h

```
#ifndef __CLIENTE_H__
#define __CLIENTE_H__

#include <Ice/Ice.h>
#include <list>
#include <iostream>
#include <queue>
#include <deque>

#define ERROR -1
#define SI 1
#define NO 0
#define HIERBA 1
#define AGUA 2
#define BARRO 3
#define MURAYA 4
#define HOYO 5
#define ZANJA 6
#define BANDERA 7
#define BARCA 8
#define HACHA 9
#define ZUMO 10
#define PALA 11
#define BOSQUE 12
#define LINEAS_CLEAR 40
#define MAX_JUGADORES 16
#define COLA_FIFO 1
#define COLA_LIFO 2
#define COLA_ORDENADA 3
#define INFINITO 1000
#define MAX_REPETIR_CASILLA 2
#define MAXMEM_DEFAULT 262144
#define MAXLINEA 255
#define MAX_COSTO 1000
#define PESO_ENERGIA_COLINAS 0.1
#define PESO_DISTANCIA_COLINAS 0.3
#define PESO_NBANDERAS_COLINAS 0.6
#define FACTOR_CORRECCION_DISTANCIA 1.25
#define LIMITE_LATERAL_COLINAS 100
#define CAMBIO_ALGORITMO1 999
#define CAMBIO_ALGORITMO2 9999
#define FACTOR_CORRECION_DISTANCIA_AGUA 6
#define INFINITO 1000
#define NACCIONES 6
#define PESO_MAX_EVAL_PARA_BANDERAS 0.5
#define PESO_MAX_EVAL_PARA_ENERGIA 0.1
#define PESO_MAX_EVAL_PARA_BARCA 0.05
#define PESO_MAX_EVAL_PARA_PALA 0.02
#define PESO_MAX_EVAL_PARA_PROFUNDIDAD 0.2
#define PESO_MAX_EVAL_PARA_HACHA 0.03
#define PESO_MAX_EVAL_CONTRARIO 0.1
/*
unsigned char  8 bits  0               a  255
char           8 bits  -128            a  127
short int     16 bits  -32,768         a  32,767
unsigned int  32 bits  0               a  4,294,967,295
int           32 bits  -2,147,483,648  a  2,147,483,647
unsigned long 32 bits  0               a  4,294,967,295
enum          16 bits  -2,147,483,648  a  2,147,483,647
long          32 bits  -2,147,483,648  a  2,147,483,647
float         32 bits  3.4*10-38       a  3.4*10+38     (6 dec)
double        64 bits  1.7*10-308      a  1.7*10+308    (15 dec)
long double   80 bits  3.4*10-4932     a  1.1*10+4932
*/
typedef signed char  sint8;
typedef signed char  int8;
typedef signed char  uint8;
typedef signed short sint16;
typedef signed short int16;
typedef signed short uint16;
typedef signed int   sint32;
typedef signed int   int32;
typedef signed int   uint32;
```

```
typedef signed long int sint64;
typedef signed long int int64;
typedef signed long int uint64;

#define int8_size 1
#define int16_size 2
#define int32_size 4
#define int64_size 8
#define obj_ptr_size 8

#endif
```

6.6. Client.cpp

```
#include <Ice/Ice.h>
#include "Client.h"
#include "Clases.h"
#include "Practica.h"
#include "Colores.h"
#include "Problema.h"
#include <math.h>
#include <sys/time.h>
#include <pthread.h>
#include <semaphore.h>
#include <sys/signal.h>

bool devuelto; int val,i,idUsuario,profundidad_actual=2,sleep_time=1,do_sleep=0;
int8 njugadores=0,nbanderas=0;
int16 ncasillas=0;
pthread_t hilo;
pthread_mutex_t mutex;
vector<int16> banderas_numero;
int tiempo_max=0,delta_t=2000;
using namespace std;
using namespace Practica;
Nodo *nodo_solucion=NULL,*nodo_actual,*nodo_de_reserva;
Practica::Mapa tablero; bool a_la_desesperada=false;

void *ejecucion_hilo(void *p){
        Nodo *n;
        while (1) {
                n=Busqueda_Alfa_Beta(tablero,nodo_actual,profundidad_actual++);
                pthread_mutex_lock (&mutex);
                        nodo_solucion=n;
                pthread_mutex_unlock (&mutex);
        }
}

class Client : virtual public Ice::Application{
  void printHelp(){
    cout << "\nEjecucion del cliente para las practicas de IA: " << endl;
    cout << "- Para jugar una partida individual ejecutar dos instancias del programa con la
linea: ./Client 1 --Ice.Config=(Ruta Archivo .config)\n" << endl;
    cout << "- Para jugar una competicion ejecutar una instancia del programa con la linea: ./Client
2 --Ice.Config=(Ruta Archivo .config)\n  Esperar a que el oponente se una a la partida.\n\n" <<
endl;
  }

  void jugarPartida(string dni, Practica::PartidaPrx partida){
    FILE *fich; char cadaux[MAXLINEA];

    if(!(fich = fopen ("Configuracion", "r"))){
            cout << endl << "No se encuentra el archivo de configuracion" << endl;
    }
    fgets(cadaux, MAXLINEA, fich); sscanf(&cadaux[0], "PROFUNDIDAD=%d",&profundidad_actual);
    fgets(cadaux, MAXLINEA, fich); sscanf(&cadaux[0], "SLEEP=%d",&do_sleep);
    fgets(cadaux, MAXLINEA, fich); sscanf(&cadaux[0], "TIME=%d",&sleep_time);
    fclose (fich);

    bool jugando = true; srand(time(0));
    tablero = partida->obtenerMapa(dni);
    tiempo_max=tablero.tiempo-delta_t;
    idUsuario=tablero.idUsuario;

    for (i=0;i<ncasillas;i++){
       if (tablero.casillas[i]==BANDERA){
```

```
                banderas_numero.push_back(i); nbanderas++;
        }
    }

    cout << "Soy el jugador: " << idUsuario << endl;

    //creamos el nodo raiz
    njugadores=(int8)(tablero.jugadores.size()/2);
    ncasillas=(int16)tablero.dimx*tablero.dimy;

    vector<Jugadores*> jugadores;
    vector<Casilla*> casillas;

    //setear jugadores
    if (idUsuario==1) {
        for (i=0;i<njugadores*2;i++){
                jugadores.push_back(new
Jugadores(i+1,tablero.jugadores[i].casilla,tablero.jugadores[i].energia,0,0,0));
        }
    } else {
        for (i=0;i<njugadores;i++){
                jugadores.push_back(new
Jugadores(i+1,tablero.jugadores[i+njugadores].casilla,tablero.jugadores[i+njugadores].energia,0,0,0)
);
        }
        for (i=njugadores;i<njugadores*2;i++){
                jugadores.push_back(new Jugadores(i+1,tablero.jugadores[i-
njugadores].casilla,tablero.jugadores[i-njugadores].energia,0,0,0));
        }
    }

    //crear estado inicial
    Estado *estado_inicial = new Estado(0,&jugadores,&casillas);

    //crear nodo inicial
    Nodo *raiz = new Nodo(estado_inicial,NULL,0,0,0,0,+INFINITO);
    nodo_actual=raiz;

    while(jugando){
        bool moviendo = true;
        Practica::InfoJugada info = partida->pedirTurno(tablero.idUsuario);

        if(info.resultado != 0){
                if(info.resultado == 1){
                  cout << "¡Has GANADO!" << endl;
                 nodo_actual->Print();
                  jugando = false;
                  moviendo = false;
                }
         else{
                  cout << "¡Has PERDIDO!" << endl;
                 nodo_actual->Print();
                  jugando = false;
                  moviendo = false;
         }

        //actualizar la información del contrario
        if (info.mov.idJugador!=-1) {
                if (idUsuario==1) {
                        Estado *e=nodo_actual->estado;
                        e->Movimiento(info.mov.idJugador-1,info.mov.mov);
                } else {
                        Estado *e=nodo_actual->estado;
                        e->Movimiento(info.mov.idJugador-1+njugadores,info.mov.mov);
                }
        }
         break;
      }

      while(moviendo){
                if (!a_la_desesperada) {
                        profundidad_actual=2;
                        //crear hilo
                        pthread_create(&hilo, NULL, (void *)&ejecucion_hilo,(void *)NULL);
                        //dormir
                        sleep(tablero.tiempo-sleep_time);
                        //destruir hilo
                        pthread_destroy(&hilo);
                        //coger ultima solucion
```

```
                        nodo_de_reserva=nodo_actual;
                        pthread_mutex_lock (&mutex);
                                nodo_actual=nodo_solucion;
                        pthread_mutex_unlock (&mutex);
                } else {
                        nodo_actual=NULL;
                }

                try{
                   if (nodo_actual!=NULL) {
                        cout << endl << "JUGADA-> Jugador: " << (int)nodo_actual->jugador << "
Movimiento: " << (int)nodo_actual->movimiento << "->";
                        cout << " a profundidad " << (int) profundidad_actual;
                        bool devuelto = partida->jugada(tablero.idUsuario, nodo_actual->jugador,
nodo_actual->movimiento, info.token);
                        cout << " Movimiento realizado: " << devuelto << endl;
                        moviendo = false;
                    } else {
                        cout << endl << "Nos hemos quedado sin sucesores"; fflush(stdout);
                        cout << endl << "A la desesperada";
                        nodo_actual=nodo_de_reserva;
                        nodo_actual->Print();
                        for (int i=0;i<njugadores;i++){
                              Estado *e=nodo_actual->estado;
                              (e->jugadores[i])->energia=20;
                        }
                        vector<Nodo*> *sucesores=Sucesores(nodo_actual);
                        nodo_actual=(*sucesores)[rand()%sucesores->size()];

                        cout << "Introduzca de nuevo el movimiento para salir del paso" << endl;
                        bool devuelto = partida->jugada(tablero.idUsuario, nodo_actual->jugador,
nodo_actual->movimiento, info.token);
                        cout << " Movimiento realizado: " << devuelto << endl;
                        moviendo = false;
                        for (unsigned int i=0;i<sucesores->size();i++){
                              delete (*sucesores)[i];
                        } delete sucesores;
                    }
                }
                catch(Practica::TokenIncorrectoError e){
                  cout << e.reason << endl;
                  cout << "El temporizador ha expirado. Has PERDIDO la partida" << endl;
                  jugando = false; moviendo = false;
                }
                catch(Practica::MovimientoIncorrectoError e){
                  cout << e.reason << endl;
                  cout << "Ha introducido un movimiento no valido. Introduzca de nuevo el movimiento.
" << endl;
                  nodo_actual->Print();
                  vector<Nodo*> *sucesores=Sucesores(nodo_actual);
                  nodo_actual=(*sucesores)[rand()%sucesores->size()];

                  cout << "Introduzca de nuevo el movimiento para salir del paso" << endl;
                  bool devuelto = partida->jugada(tablero.idUsuario, nodo_actual->jugador,
nodo_actual->movimiento, info.token);
                  cout << " Movimiento realizado: " << devuelto << endl;
                  moviendo = false;
                  for (unsigned int i=0;i<sucesores->size();i++){
                        delete (*sucesores)[i];
                  } delete sucesores;
                }
      }
    }
  }

  virtual int run(int argc, char* argv[]){
    string dni = "5698168";
    string password = "blacksun";

    if(argc == 1 || argc > 2) { printHelp(); }
    else{
      Ice::ObjectPrx base = communicator()->stringToProxy("AutenticacionObject");
      Practica::AutenticacionPrx autenticacion = Practica::AutenticacionPrx::checkedCast(base);

      if(!autenticacion){
        cout << " Error obteniendo el proxy de autenticacion" << endl;
        return 1;
      }
      Practica::PartidaPrx partida = NULL;
```

```
      try{
        partida = autenticacion->login(dni, password);
      }

      catch(Practica::PasswordIncorrectaError e){
        cout << e.reason << endl;
      }
      catch(Practica::UsuarioIncorrectoError e){
        cout << e.reason << endl;
      }
      catch(Practica::NoExisteContrincanteError e){
        cout << e.reason << endl;
      }
      catch(Practica::NoExistePartidaError e){
        cout << e.reason << endl;
      }

      if(!strcmp(argv[1], "1") && partida != NULL){
        jugarPartida(dni, partida);
        autenticacion->finalizarPartida(dni, password);
      }

      else if(!strcmp(argv[1], "2") && partida != NULL){
        jugarPartida(dni, partida);
        cout << "RECUPERANDO SEGUNDA PARTIDA" << endl;
        sleep(5);
        jugarPartida(dni, partida);
        autenticacion->finalizarPartida(dni, password);
      }

      else if(partida != NULL){
        cout << "ERROR: El argumento introducido no es valido" << endl;
        printHelp();
      }
    }

    return 0;
  }
};

int main(int argc, char* argv[]){
  Client c;
  c.main(argc, argv);
}
```

6.7. Problema.h

```
#ifndef __PROBLEMA_H_
#define __PROBLEMA_H_

#include "Clases.h"
#include "Client.h"
#include <Ice/Ice.h>
#include <Practica.h>
#include <list>
#include <iostream>
#include <queue>
#include <deque>
#include <vector>
#include "Colores.h"
#include <time.h>
#include <memory>

using namespace Practica;
using namespace std;

float MIN(float a, float b);
float MAX(float a, float b);
double medicion_tiempo();
float distancia_casillas(int16 c1,int16 c2,bool tile);
int16 SiguienteCasilla(int16 casilla,int8 mov);
float Eval(Nodo *nodo);
Nodo *Busqueda_Alfa_Beta(Mapa map,Nodo *nodo,int profundidad_max);
float Max_ValorAB(Nodo *nodo,float alfa,float beta,int profundidad_actual,vector<Nodo*> *sucesores);
float Min_ValorAB(Nodo *nodo,float alfa,float beta,int profundidad_actual,vector<Nodo*> *sucesores);
vector<Nodo*> *Sucesores(Nodo *nodo);
float alphabeta(Nodo *nodo, int depth, float alfa, float beta);
#endif
```

6.8. Problema.cpp

```
#include "Problema.h"
#include "Clases.h"
#include <math.h>
/*****************************************************************************************
VARIABLES GLOBALES
*****************************************************************************************/
Mapa mapa;
Frontera *frontera;
extern int8 njugadores,nbanderas;
extern int16 ncasillas;
int nodos_creados=0;
extern double tiempo_max;
extern vector<int16> banderas_numero;
int painprimir=0;
int maxbanderas=0;
extern int idUsuario;
/*****************************************************************************************
MACROS Y ALGUNAS FUNCIONES
*****************************************************************************************/
#define round(x) (x<0?ceil((x)-0.5):floor((x)+0.5))

float MIN(float a, float b) {
	if (a<b) { return a; }
	else { return b; }
}

float MAX(float a,float b) {
	if (a>b) { return a; }
	else { return b; }
}

double medicion_tiempo(){
	struct timeval x; struct timezone y; gettimeofday(&x,&y);
	return x.tv_sec+x.tv_usec/1000000.0;
}
/*****************************************************************************************
FUNCIONES HEURISTICAS DE A* Y ASCENSIÓN COLINAS
*****************************************************************************************/
float distancia_casillas(int16 c1,int16 c2,bool tile){
	if (c1==c2) { return 0; }
	else if (abs(c1-c2)==1) { return 1; }
	else {
		int16 fila_c1,columna_c1,fila_c2,columna_c2,dif_x,dif_y;
		fila_c1=c1/mapa.dimx;
		if (c1%mapa.dimx!=0){ fila_c1++; }
		if (c1>mapa.dimx){
			columna_c1=c1-((fila_c1-1)*mapa.dimx);
			if (columna_c1==0) { columna_c1=mapa.dimx; }
		} else { columna_c1=c1; }

		fila_c2=c2/mapa.dimx;
		if (c2%mapa.dimx!=0){ fila_c2++; }
		if (c2>mapa.dimx){
			columna_c2=c2-((fila_c2-1)*mapa.dimx);
			if (columna_c2==0) { columna_c2=mapa.dimx; }
		} else { columna_c2=c2; }

		dif_x=abs(fila_c1-fila_c2);
		if (tile) {
			if (fila_c1>fila_c2) {
				if ((mapa.dimy-fila_c1+fila_c2-1) <dif_x ) {
					dif_x=(mapa.dimy-fila_c1+fila_c2-1);
				}
			}
			if (fila_c1<fila_c2) {
				if ((fila_c1+mapa.dimy-fila_c2-1) <dif_x) {
					dif_x=(fila_c1+mapa.dimy-fila_c2-1);
				}
			}
		}

		dif_y=abs(columna_c1-columna_c2);
		if (tile) {
			if (columna_c1>columna_c2) {
```

```
                        if ((mapa.dimx-columna_c1+columna_c2-1) <dif_x ) {
                                dif_y=(mapa.dimx-columna_c1+columna_c2-1);
                        }
                }
                if (columna_c1<columna_c2) {
                        if ((columna_c1+mapa.dimx-columna_c2-1) <dif_x ) {
                                dif_y=(columna_c1+mapa.dimx-columna_c2-1);
                        }
                }
        }

        float hip=(sqrt( (dif_x*dif_x) + (dif_y*dif_y) ));
        return (round(hip));
    }
}
/*******************************************************************************************
CALCULO DE SUCESORES
*******************************************************************************************/
vector<Nodo*> *Sucesores(Nodo *nodo){
    vector<Nodo*> *s=new vector<Nodo*>;
    int8 i,j,veces,accion[NACCIONES],jugador[MAX_JUGADORES],k,swap,p1,p2;
    sint8 resultado=0,nodos_internos=0;
    int16 energia_antes,energia_despues,delta_energia;
    sint16 costo;

    //aleatoriedad a la hora de cual meter primero en la frontera
    for (i=0;i<NACCIONES;i++){ accion[i]=i+1; }
    for (i=0;i<njugadores;i++) { jugador[i]=i; }
    for (i=0;i<NACCIONES;i++){
        p1=rand()%NACCIONES; p2=rand()%NACCIONES;
        swap=accion[p1]; accion[p1]=accion[p2]; accion[p2]=swap;
    }
    for (i=0;i<njugadores;i++){
        p1=rand()%njugadores; p2=rand()%njugadores;
        swap=jugador[p1]; jugador[p1]=jugador[p2]; jugador[p2]=swap;
    }

    for (i=0;i<NACCIONES;i++){ //n_aciones=aciones*njugadores
        for (j=0;j<njugadores;j++){
            //no permitir acciones contrarias al padre
            if ((jugador[j]==nodo->jugador) && ((accion[i]==nodo->movimiento+3)%6) &&
(accion[i]!=7)){ continue; }
            //copiar el estado del padre y modificarlo a partir de la copia
            Estado *e;
            try {
                e=new Estado(nodo->estado->banderas,&nodo->estado->jugadores,&nodo-
>estado->casillas);
            } catch (std::bad_alloc) {
                cout << endl << "No hay disponible memoria para crear más instancias de
Estado";
                exit(1);
            }
            energia_antes=(e->jugadores[jugador[j]]->energia);

            //resultado de hacer que el jugador j haga el movimiento i

            resultado=e->Movimiento(jugador[j],accion[i]);

            energia_despues=(e->jugadores[jugador[j]]->energia);

            //Calculo del costo, basado en la energia del jugador
            delta_energia = energia_antes-energia_despues;

            if ( delta_energia < 0 ) { costo=nodo->costo; }
            else { costo=(nodo->costo)+delta_energia; }

            //comprobación de que no hemos pasado ya por esa casilla
            Nodo *puntero_padre=nodo; k=nodo->profundidad; veces=0;
            while ((k--)>0){ //hasta que nos encontremos con la raiz subimos comprobando
                if (puntero_padre->jugador==j+1){
                    if ((puntero_padre->estado->jugadores[j]->casilla)==(e-
>jugadores[jugador[j]]->casilla)){
                        veces++; if (veces > MAX_REPETIR_CASILLA)
{ resultado=ERROR; break; }
                    }
                }
                puntero_padre=nodo->nodo_padre;
            }
```

```
                        //si es un movimiento que se puede hacer entonces es un nodo hijo y se agrega
a sucesores
                        if (resultado!=ERROR){
                                nodos_creados++; nodos_internos++;
                                e->banderas=e->banderas+resultado;
                                Nodo *hijo;
                                try {
                                        if (idUsuario==1) {
                                                hijo=new Nodo(e,nodo,jugador[j]+1,accion[i],
(sint16)costo,(nodo->profundidad)+1,+INFINITO);
                                        } else {
                                                hijo=new Nodo(e,nodo,jugador[j]+1+njugadores,accion[i],
(sint16)costo,(nodo->profundidad)+1,+INFINITO);
                                        }
                                } catch (std::bad_alloc) {
                                        cout << endl << "No hay disponible memoria para crear más
instancias de Nodo";
                                        exit(1);
                                }
                                //Eval(hijo);
                                s->push_back(hijo);
                        } else { delete e; }
                }
        }
        if (s->size()>0) { return s; }
        else { delete s; return NULL; }
}

/*********************************************************************************************
BUSQUEDA ALFA-BETA
*********************************************************************************************/
Nodo *Busqueda_Alfa_Beta(Mapa map,Nodo *nodo,int profundidad_max){
        Nodo *accion=NULL; mapa=map; unsigned int no_borrar=0; float minV;
        vector<Nodo*> *sucesores=Sucesores(nodo);
        vector<Nodo*> *candidatos=new vector<Nodo*>;
        if (sucesores!=NULL) {
                float v=Max_ValorAB(nodo,-INFINITO,+INFINITO,profundidad_max,sucesores);
                for (unsigned int i=0;i<sucesores->size();i++) {
                        if (v==((*sucesores)[i])->v){ candidatos->push_back((*sucesores)[i]); }
                        else { delete (*sucesores)[i]; }
                }
                delete sucesores;

                //de los nodos que tenemos con el mismo valor minimax elegiremos el que mejor
valoracion propia tenga
                //es como si aplicamos colinas a los estados con el mismo valor minimax
                minV=+INFINITO;
                for (unsigned int i=0;i<candidatos->size();i++) {
                        Eval((*candidatos)[i]);
                        if (((*candidatos)[i])->v<minV) {
                                accion=(*candidatos)[i];
                                no_borrar=i;
                        }
                }

                for (unsigned int i=0;i<candidatos->size();i++) {
                        if (i!=no_borrar){ delete (*candidatos)[i];}
                } delete candidatos;
        }
        return accion;
}
float Max_ValorAB(Nodo *nodo,float alfa,float beta,int profundidad_actual,vector<Nodo*> *sucesores){
        nodo->v=-INFINITO; vector<Nodo*> *sucesores_next; bool clear=false;
        if ((profundidad_actual==1) || (sucesores==NULL)) { Eval(nodo); }
        else {
                for (unsigned int i=0;i<sucesores->size();i++) {
                        sucesores_next=Sucesores((*sucesores)[i]);
                        if (sucesores_next!=NULL) {
                                nodo->v=MAX(nodo->v,Min_ValorAB(((*sucesores)
[i]),alfa,beta,profundidad_actual-1,sucesores_next));
                                if (nodo->v>=beta) { clear=true; break; }
                                alfa=MAX(alfa,nodo->v);
                                for (unsigned int i=0;i<sucesores_next->size();i++) {
                                        delete (*sucesores_next)[i];
                                } delete sucesores_next;
                        }
                }
        }
        if (clear) {
```

```
                for (unsigned int i=0;i<sucesores_next->size();i++) {
                        delete (*sucesores_next)[i];
                } delete sucesores_next;
        }
        return nodo->v;
}
float Min_ValorAB(Nodo *nodo,float alfa,float beta,int profundidad_actual,vector<Nodo*> *sucesores){
        nodo->v=+INFINITO; vector<Nodo*> *sucesores_next; bool clear=false;
        if ((profundidad_actual==1) || (sucesores==NULL)) { Eval(nodo); }
        else {
                for (unsigned int i=0;i<sucesores->size();i++) {
                        sucesores_next=Sucesores((*sucesores)[i]);
                        if (sucesores_next!=NULL) {
                                nodo->v=MIN(nodo->v,Max_ValorAB(((*sucesores)
[i]),alfa,beta,profundidad_actual-1,sucesores_next));
                                if (nodo->v<=alfa) { clear=true; break; }
                                beta=MIN(beta,nodo->v);
                                for (unsigned int i=0;i<sucesores_next->size();i++) {
                                        delete (*sucesores_next)[i];
                                } delete sucesores_next;
                        }
                }
        }
        if (clear) {
                for (unsigned int i=0;i<sucesores_next->size();i++) {
                        delete (*sucesores_next)[i];
                } delete sucesores_next;
        }
        return nodo->v;
}

/*****************************************************************************************
FUNCIÓN DE EVALUACIÓN
*****************************************************************************************/
float Eval(Nodo *nodo){
        float value1=0,value2=0,value3=0,value4=0,distancia_minima,distancia;
        unsigned int bandera_eliminar=0,jugador_cambiar=0;
        vector<int16> *b=new vector<int16>; vector<int16> *j=new vector<int16>;

        for (int i=0;i<njugadores;i++){ //añado los jugadores
                j->push_back(nodo->estado->jugadores[i]->casilla);
        }

        for (int i=0;i<nbanderas;i++){ //añado las banderas
                if (!(nodo->estado->EstaCogidaBandera(banderas_numero[i]))){
                        b->push_back(banderas_numero[i]);
                }
        }

        while ( b->size() > 0){
                distancia_minima=INFINITO; bandera_eliminar=0;
                for (unsigned int i=0;i<j->size();i++){
                        for (unsigned int k=0;k<b->size();k++){
                                distancia=distancia_casillas((*j)[i],(*b)[k],true);
                                if (distancia<distancia_minima){
                                        distancia_minima=distancia;  bandera_eliminar=k;
jugador_cambiar=i;
                                }
                        }
                }

                (*j)[jugador_cambiar]=(*b)[bandera_eliminar];
                b->erase(b->begin()+bandera_eliminar);
                value1+=distancia_minima;
        }

        //preferencia por estados con casillas de jugador de menos costo
        for (int i=0;i<njugadores;i++){ //añado los jugadores
                switch (nodo->estado->jugadores[i]->casilla) {
                        case HIERBA:   value2+=(1/10); break;
                        case AGUA:     if (nodo->estado->jugadores[i]->barca>0) {value2+=(3/10);}
                                       else { value2+=(6/10); } break;
                        case BARRO:    value2+=(2/10); break;
                        case HOYO:     value2+=(4/10); break;
                        case ZANJA:    value2+=(6/10); break;
                        case BOSQUE:   if (nodo->estado->jugadores[i]->hacha>0) { value2+=(4/10);}
                                       else { value2+=(8/10); } break;
                }
        }
```

```
        b->clear(); j->clear();
        value3 =  1 - ((float)(nodo->estado->banderas)+1)/((float)(nbanderas+1));
        value4=(nodo->profundidad)*0.1;

        nodo->v=-(value1 + value2 + value3 + value4);
        delete b; delete j;
        return nodo->v;
}
/*****************************************************************************************
CALCULA LA SIGUIENTE CASILLA DADA LA ACTUAL Y UN MOVIMIENTO
*****************************************************************************************/
int16 SiguienteCasilla(int16 casilla,int8 mov){
        if (mov!=7){ //Se esta haciendo un desplazamiento
                int16 siguiente_casilla,fila,columna;

                fila=casilla/mapa.dimx;
                if (casilla%mapa.dimx!=0){ fila++; }

                if (casilla>mapa.dimx){
                        columna=casilla-((fila-1)*mapa.dimx);
                        if (columna==0) { columna=mapa.dimx; }
                } else { columna=casilla; }

                if (casilla%2==1){ //casillas impares empezando por el 1
                                switch(mov){
                                case 1:
                                                if (fila==1) { siguiente_casilla=(mapa.dimx*mapa.dimy)-
(mapa.dimx-casilla); }
                                                else { siguiente_casilla=casilla-mapa.dimx; } break;
                                case 2:
                                                if (casilla==mapa.dimx)
{ siguiente_casilla=(mapa.dimx*mapa.dimy)-mapa.dimx; }
                                                else if (fila==1)
{ siguiente_casilla=(mapa.dimx*mapa.dimy)-(mapa.dimx-casilla)+1; }
                                                else { siguiente_casilla=casilla-(mapa.dimx-1); } break;
                                case 3:
                                                siguiente_casilla=casilla+1; break;
                                case 4:
                                                if (fila==mapa.dimy) { siguiente_casilla=mapa.dimx-
((mapa.dimx*mapa.dimy)-casilla); }
                                                else { siguiente_casilla=casilla+mapa.dimx; } break;
                                case 5:
                                                if (casilla==(mapa.dimx*mapa.dimy)-mapa.dimx+1)
{ siguiente_casilla=(mapa.dimx*mapa.dimy); }
                                                if (columna==1){ siguiente_casilla=casilla+
(mapa.dimx)-1; }
                                                else { siguiente_casilla=casilla-1; } break;
                                case 6:
                                                if (casilla==1)
{ siguiente_casilla=(mapa.dimx*mapa.dimy); }
                                                else if (columna==1){ siguiente_casilla=casilla-1; }
                                                else if (fila==1)
{ siguiente_casilla=(mapa.dimx*mapa.dimy)-(mapa.dimx-casilla)-1; }
                                                else { siguiente_casilla=casilla-(mapa.dimx+1);     }
break;
                                }
                } else { //casillas pares empezando por el 2
                                switch(mov){
                                case 1:
                                                if (fila==1) { siguiente_casilla=(mapa.dimx*mapa.dimy)-
(mapa.dimx-casilla); }
                                                else { siguiente_casilla=casilla-mapa.dimx; } break;
                                case 2:
                                                if (columna==mapa.dimx){ siguiente_casilla=casilla-
mapa.dimx+1; }
                                                else { siguiente_casilla=casilla+1; } break;
                                case 3:
                                                if (casilla==(mapa.dimx*mapa.dimy))
{ siguiente_casilla=1; }
                                                else if (columna==mapa.dimx)
{ siguiente_casilla=casilla+1; }
                                                else if (fila==mapa.dimy){ siguiente_casilla=mapa.dimx-
((mapa.dimx*mapa.dimy)-casilla)+1; }
                                                else { siguiente_casilla=casilla+(mapa.dimx+1);     }
break;
                                case 4:
                                                if (fila==mapa.dimy) { siguiente_casilla=mapa.dimx-
((mapa.dimx*mapa.dimy)-casilla); }
```

```
                                        else { siguiente_casilla=casilla+mapa.dimx; } break;
                                case 5:
                                        if (fila==mapa.dimy){ siguiente_casilla=mapa.dimx-
((mapa.dimx*mapa.dimy)-casilla)-1; }
                                        else { siguiente_casilla=casilla+(mapa.dimx-1);     }
break;
                                case 6:
                                        siguiente_casilla=casilla-1; break;
                                }
                }
                return siguiente_casilla;
        } else { //Se esta cavando
                        int16 i=mapa.casillas[casilla];
                        if (i==HIERBA || i==BARRO || i==HOYO){       return casilla;       }
                        else { return ERROR; }
        }
}
```

6.9. Colores.h

```
#ifndef __COLORES_H_
#define __COLORES_H_

#include <stdio.h>

void color_normal();
void color_negro();
void color_cyan();
void color_cyan_claro();
void color_rojo();
void color_rojo_claro();
void color_verde();
void color_verde_claro();
void color_amarillo();
void color_marron();
void color_gris_oscuro();
void color_gris_claro();
void color_azul();
void color_azul_claro();
void color_purpura();
void color_purpura_claro();

#endif
```

6.10. Colores.cpp

```
#include "Colores.h"

void color_normal()        { putc(27,stdout); fprintf(stdout,"[0;00m"); }
void color_negro()         { putc(27,stdout); fprintf(stdout,"[0;30m"); }
void color_cyan()          { putc(27,stdout); fprintf(stdout,"[0;36m"); }
void color_cyan_claro()    { putc(27,stdout); fprintf(stdout,"[1;36m"); }
void color_rojo()          { putc(27,stdout); fprintf(stdout,"[0;31m"); }
void color_rojo_claro()    { putc(27,stdout); fprintf(stdout,"[1;31m"); }
void color_verde()         { putc(27,stdout); fprintf(stdout,"[0;32m"); }
void color_verde_claro()   { putc(27,stdout); fprintf(stdout,"[1;32m"); }
void color_amarillo()      { putc(27,stdout); fprintf(stdout,"[1;33m"); }
void color_marron()        { putc(27,stdout); fprintf(stdout,"[0;33m"); }
void color_gris_oscuro()   { putc(27,stdout); fprintf(stdout,"[1;30m"); }
void color_gris_claro()    { putc(27,stdout); fprintf(stdout,"[0;37m"); }
void color_azul()          { putc(27,stdout); fprintf(stdout,"[0;34m"); }
void color_azul_claro()    { putc(27,stdout); fprintf(stdout,"[1;34m"); }
void color_purpura()       { putc(27,stdout); fprintf(stdout,"[0;35m"); }
void color_purpura_claro(){ putc(27,stdout); fprintf(stdout,"[1;35m"); }
```